# Homework #1

**( Due: February 14 )**

GAUSSIAN$(c, 1, n)$
(The input $c[1 \ldots n, 1 \ldots n]$ is an $n \times n$ matrix with all diagonal entries set to nonzero values.)

1. ***for*** $k \leftarrow 1$ ***to*** $n$ ***do***
2. ***for*** $i \leftarrow 1$ ***to*** $n$ ***do***
3. ***for*** $j \leftarrow 1$ ***to*** $n$ ***do***
4. ***if*** $(k \leq n-2) \ \wedge \ (k < i < n) \ \wedge \ (k < j)$ ***then*** $c[i,j] \leftarrow c[i,j] - \frac{c[i,k]}{c[k,k]} \times c[k,j]$

Figure 1: The first phase of Gaussian elimination without pivoting.

**Task 1.** [ **50 Points** ] Implement the first phase of Gaussian elimination without pivoting.

(a) [ **5 Points** ] Try parallelizing the naïve version (with triply nested `for` loops as given in Figure 1) using `cilk_for`. Explain your findings.

(b) [ **15 Points** ] Parallelize based on the divide-and-conquer framework given in the appendix, and optimize your implementation as much as possible (e.g., optimize the base case size). For simplicity use only values of $n$ that are powers of 2.

(c) [ **15 Points** ] Analyze the parallelism of the divide-and-conquer algorithm. (Hint: First compute $T_1$ and $T_\infty$ for function D. Use those to compute $T_1$ and $T_\infty$ for B and C. Then analyze A.)

(d) [ **5 Points** ] Find the largest value of $n$ (say, $n_{max}$) for which your implementation in part (b) terminates in less than 15 minutes on a single core. For each $n = 2^k \leq n_{max}$ (where $k \geq 4$), generate a Cilkview scalability plot.

(e) [ **5 Points** ] For the same set of $n$ values as in part (d) plot the speedup factors achieved by the parallel implementation in part (b) w.r.t. the serial loop implementation as you vary the number of cores. In order to fix the number of cores (or threads) used pass the command line parameter `-cilk_set_worker_count=k` to your program, where $k$ is the number of threads to use.

(f) [ **5 Points** ] Use `PAPI` (`http://icl.cs.utk.edu/papi/`) to compare the number of L1 and L2 cache misses incurred by the parallel implementation in part (b) when run on a single core, and the serial loop implementation as you vary $n$. In order to load `PAPI` use "module load papi/3.6.0" on `Ranger`, and "module load papi/4.1.2.1" on `Loanstar`.

```
FW-APSP(c, 1, n)
(The input c[1...n, 1...n] is an n × n matrix with all diagonal entries set to zero and the remaining entries set to positive values.)
1. for k ← 1 to n do
2.     for i ← 1 to n do
3.         for j ← 1 to n do
4.             c[i, j] ← min {c[i, j], c[i, k] + c[k, j]}
```

Figure 2: Floyd-Warshall's APSP (all-pairs shortest paths) with nonnegative edge weights.

**Task 2.** [ **30 Points** ] Repeat Task 1 (without parts (a) and (c)) for Floyd-Warshall's APSP (Figure 2).

```
SIMPLE-LJ(p, n)
(The input is an array p[1...n] of n points in 3-space. The function returns a Lennard-Jones-like potential value computed for those points.)
1. E ← 0
2. for i ← 1 to n do
3.     for j ← 1 to n do
4.         if i ≠ j then
5.             d ← √((p[i].x − p[j].x)² + (p[i].y − p[j].y)² + (p[i].z − p[j].z)²)
6.             E ← E + (1/d^12 − 1/d^6)
7. return E
```

Figure 3: Simple Lennard-Jones Potential.

**Task 3.** [ **10 Points** ] Parallelize the computation in Figure 3. Plot the speedup factors achieved w.r.t. the single-core execution of the parallel version as well as the serial version as the number of cores is varied. Use $n = 10^k$, $2 \leq k \leq 5$.

**Task 4.** [ **20 Points** ] Implement parallel mergesort with and without parallel merge. Generate Cilkview scalability plots for $n = 10^k$, $3 \leq k \leq 7$.

# APPENDIX: A Common Algorithmic Framework for Gaussian Elimination without Pivoting and Floyd-Warshall's APSP

G($c,\ n,\ f,\ \Sigma_G$)

(Input $c[1\ldots n, 1\ldots n]$ is an $n\times n$ matrix, $f(\cdot,\cdot,\cdot,\cdot)$ is an arbitrary problem-specific function, and $\Sigma_G$ is a problem-specific set of triplets such that $c[i,j] \leftarrow f(c[i,j], c[i,k], c[k,j], c[k,k])$ is executed in line 4 if $\langle i,j,k\rangle \in \Sigma_G$.)

1. ***for*** $k \leftarrow 1$ ***to*** $n$ ***do***
2. ***for*** $i \leftarrow 1$ ***to*** $n$ ***do***
3. ***for*** $j \leftarrow 1$ ***to*** $n$ ***do***
4. ***if*** $\langle i,j,k\rangle \in \Sigma_G$ ***then*** $c[i,j] \leftarrow f(c[i,j], c[i,k], c[k,j], c[k,k])$

Figure 4: Triply nested ***for*** loops typifying code fragment with structural similarity to the computation in Gaussian elimination without pivoting.

Let $c[1\ldots n, 1\ldots n]$ be an $n \times n$ matrix with entries chosen from an arbitrary set $\mathcal{S}$, and let $f : \mathcal{S}\times\mathcal{S}\times\mathcal{S}\times\mathcal{S} \rightarrow \mathcal{S}$ be an arbitrary function. In Figure 4, the algorithm G modifies $c$ by applying a given set of updates of the form $c[i,j] \leftarrow f(c[i,j], c[i,k], c[k,j], c[k,k])$, where $i,j,k \in [1,n]$; here $\langle i,j,k\rangle$ $(1 \le i,j,k \le n)$ denotes an update of the form $c[i,j] \leftarrow f(c[i,j], c[i,k], c[k,j], c[k,k])$, and we let $\Sigma_G$ denote the set of such updates that the algorithm needs to perform.

Many well-known problems can be solved using the construct in Figure 4, including all-pairs shortest paths, LU decomposition, and Gaussian elimination without pivoting. Observe that for the first phase of Gaussian elimination without pivoting given in Figure 1, $f(x,u,v,w) = x - \frac{u}{w} \times v$ and $\Sigma_G = \{\langle i,j,k\rangle : (1 \le k \le n-2)\ \wedge\ (k < i < n)\ \wedge\ (k < j \le n)\}$. For Floyd-Warshall's APSP in Figure 2, $f(x,u,v,w) = \min\{x, u+v\}$ and $\Sigma_G = \{\langle i,j,k\rangle : (1 \le i,j,k \le n)\}$

Figure 5 is a divide-and-conquer based parallel formulation of the iterative algorithm given in Figure 4. The ***parallel*** keyword in any step tells the scheduler that all function calls that follow can be executed in parallel with an implicit sync at the end.

A( $X$, $U$, $V$, $W$ )

(Each of $X$, $U$, $V$ and $W$ points to the same $2^q \times 2^q$ square submatrix of $c$ for some integer $q \geq 0$. The initial call to A is A$(c, c, c, c)$ for an $n \times n$ input matrix $c$, where $n$ is assumed to be a power of 2.)

1. ***if*** $T_{XUV} \cap \Sigma_G = \emptyset$ ***then return*** {$T_{XUV} = \{$ *updates on* $X$ *using* $(i,k) \in U$ *and* $(k,j) \in V$ $\}$, *and* $\Sigma_G$ *is the set of updates performed by the iterative code in Figure 4*}
2. ***if*** $X$ is a $1 \times 1$ matrix ***then*** $X \leftarrow f(\ X,\ U,\ V,\ W\ )$ {*base case*}

   ***else*** {*top-left, top-right, bottom-left & bottom-right quadrants of* $X$ *are denoted by* $X_{11}$, $X_{12}$, $X_{21}$ *&* $X_{22}$, *respectively.*}
3. A( $X_{11},\ U_{11},\ V_{11},\ W_{11}$ )
4. ***parallel*** : B$_1$( $X_{12},\ U_{11},\ V_{12},\ W_{11}$ ), C$_1$( $X_{21},\ U_{21},\ V_{11},\ W_{11}$ )
5. D$_1$( $X_{22},\ U_{21},\ V_{12},\ W_{11}$ )
6. A( $X_{22},\ U_{22},\ V_{22},\ W_{22}$ )
7. ***parallel*** : B$_2$( $X_{21},\ U_{22},\ V_{21},\ W_{22}$ ), C$_2$( $X_{12},\ U_{12},\ V_{22},\ W_{22}$ )
8. D$_4$( $X_{11},\ U_{12},\ V_{21},\ W_{22}$ )

B$_l$( $X$, $U$, $V$, $W$ ) { $l \in \{1,2\}$ }

($X \equiv V \equiv c[i_1..i_2, j_1..j_2]$ and $U \equiv W \equiv c[i_1..i_2, k_1..k_2]$, where $i_2 - i_1 = j_2 - j_1 = k_2 - k_1 = 2^q - 1$ for some integer $q \geq 0$, $[i_1, i_2] = [k_1, k_2]$ and $[j_1, j_2] \cap [k_1, k_2] = \emptyset$.)

1. ***if*** $T_{XUV} \cap \Sigma_G = \emptyset$ ***then return***
2. ***if*** $X$ is a $1 \times 1$ matrix ***then*** $X \leftarrow f(\ X,\ U,\ V,\ W\ )$

   ***else***
3. ***parallel*** : B$_l$( $X_{11},\ U_{11},\ V_{11},\ W_{11}$ )
   B$_l$( $X_{12},\ U_{11},\ V_{12},\ W_{11}$ )
4. ***parallel*** : D$_l$( $X_{21},\ U_{21},\ V_{11},\ W_{11}$ )
   D$_l$( $X_{22},\ U_{21},\ V_{12},\ W_{11}$ )
5. ***parallel*** : B$_l$( $X_{21},\ U_{22},\ V_{21},\ W_{22}$ )
   B$_l$( $X_{22},\ U_{22},\ V_{22},\ W_{22}$ )
6. ***parallel*** : D$_{l+2}$( $X_{11},\ U_{12},\ V_{21},\ W_{22}$ )
   D$_{l+2}$( $X_{12},\ U_{12},\ V_{22},\ W_{22}$ )

C$_l$( $X$, $U$, $V$, $W$ ) { $l \in \{1,2\}$ }

($X \equiv U \equiv c[i_1..i_2, j_1..j_2]$ and $V \equiv W \equiv c[k_1..k_2, j_1..j_2]$, where $i_2 - i_1 = j_2 - j_1 = k_2 - k_1 = 2^q - 1$ for some integer $q \geq 0$, $[j_1, j_2] = [k_1, k_2]$ and $[i_1, i_2] \cap [k_1, k_2] = \emptyset$.)

1. ***if*** $T_{XUV} \cap \Sigma_G = \emptyset$ ***then return***
2. ***if*** $X$ is a $1 \times 1$ matrix ***then*** $X \leftarrow f(\ X,\ U,\ V,\ W\ )$

   ***else***
3. ***parallel*** : C$_l$( $X_{11},\ U_{11},\ V_{11},\ W_{11}$ )
   C$_l$( $X_{21},\ U_{21},\ V_{11},\ W_{11}$ )
4. ***parallel*** : D$_{2l-1}$( $X_{12},\ U_{11},\ V_{12},\ W_{11}$ )
   D$_{2l-1}$( $X_{22},\ U_{21},\ V_{12},\ W_{11}$ )
5. ***parallel*** : C$_l$( $X_{12},\ U_{12},\ V_{22},\ W_{22}$ )
   C$_l$( $X_{22},\ U_{22},\ V_{22},\ W_{22}$ )
6. ***parallel*** : D$_{2l}$( $X_{11},\ U_{12},\ V_{21},\ W_{22}$ )
   D$_{2l}$( $X_{21},\ U_{22},\ V_{21},\ W_{22}$ )

D$_l$( $X$, $U$, $V$, $W$ ) { $l \in \{1,2,3,4\}$ }

($X \equiv c[i_1..i_2, j_1..j_2]$, $U \equiv c[i_1..i_2, k_1..k_2]$, $V \equiv c[k_1..k_2, j_1..j_2]$ and $W \equiv c[k_1..k_2, k_1..k_2]$, where $i_2 - i_1 = j_2 - j_1 = k_2 - k_1 = 2^q - 1$ for some integer $q \geq 0$, $[i_1, i_2] \cap [k_1, k_2] = \emptyset$, and $[j_1, j_2] \cap [k_1, k_2] = \emptyset$.)

1. ***if*** $T_{XUV} \cap \Sigma_G = \emptyset$ ***then return***
2. ***if*** $X$ is a $1 \times 1$ matrix ***then*** $X \leftarrow f(\ X,\ U,\ V,\ W\ )$

   ***else***
3. ***parallel*** : D$_l$( $X_{11},\ U_{11},\ V_{11},\ W_{11}$ ), D$_l$( $X_{12},\ U_{11},\ V_{12},\ W_{11}$ ), D$_l$( $X_{21},\ U_{21},\ V_{11},\ W_{11}$ ), D$_l$( $X_{22},\ U_{21},\ V_{12},\ W_{11}$ )
4. ***parallel*** : D$_l$( $X_{11},\ U_{12},\ V_{21},\ W_{22}$ ), D$_l$( $X_{12},\ U_{12},\ V_{22},\ W_{22}$ ), D$_l$( $X_{21},\ U_{21},\ V_{21},\ W_{22}$ ), D$_l$( $X_{22},\ U_{22},\ V_{22},\ W_{22}$ )

Figure 5: Multithreaded implementation of the iterative code in Figure 4. Initial call is A( $c,\ c,\ c,\ c$ ) on an $n \times n$ matrix $c$, where $n$ is a power of 2.